
# Trigger


Versão 1.2

# Trigger

Um trigger **(“gatilho”)** é um objeto programável do banco de dados associado a uma tabela. Trata-se de um procedimento que é invocado automaticamente quando um comando DML é executado na tabela, sendo executado para cada linha afetada. Desta forma, as operações que podem disparar um trigger são:

- **INSERT**
- **UPDATE**
- **DELETE**

Geralmente, os triggers são empregados para verificar integridade dos dados, fazer validação dos dados e outras operações relacionadas.

# Diferença entre Trigger e Procedimento Armazenado

Tanto os triggers quanto as stored procedures são objetos programáveis de um banco de dados. Porém, eles possuem diferenças importantes entre si, que afetam o modo como são aplicados. Algumas das principais diferenças entre trigger e procedimentos armazenados são:

- **Um Trigger é associado a uma tabela.**
- **Os triggers são armazenados no próprio banco de dados como arquivos separados.**
- **Triggers não são chamados diretamente, sendo invocados automaticamente, ao contrário dos procedimentos armazenados.**
- **Procedimentos armazenados podem trabalhar com parâmetros; Não passamos parâmetros aos triggers.**
- **Os triggers não retornam um conjunto de resultados (resultset) ao cliente chamador.**

# Prós e Contras das Triggers

Os principais pontos positivos sobre os triggers são:

- **Parte do processamento que seria executado na aplicação passa para o banco, poupando recursos da máquina cliente.**
- **Facilita a manutenção, sem que seja necessário alterar o código fonte da aplicação.**

Já contra sua utilização existem as seguintes considerações:

- **Alguém que tenha acesso não autorizado ao banco de dados poderá visualizar e alterar o processamento realizado pelos gatilhos.**
- **Requer maior conhecimento de manipulação do banco de dados (SQL) para realizar as operações internamente.**


# Aplicações dos triggers

As principais aplicações dos triggers em bancos de dados são:

- **Validação de Dados (tipos de dados, faixas de valores, etc).**
- **Rastreamento e registro de logs de atividades em tabelas.**
- **Verificação de integridade de dados e consistência**
- **Arquivamento de registros excluídos.**

# Modos de Disparo de um Trigger

Um Trigger em MySQL pode ser disparado de dois modos diferentes:

- **BEFORE –** O trigger é disparado e seu código executado ANTES da execução de cada evento – por exemplo, antes de cada inserção de registros na tabela.
- **AFTER –** O código presente no trigger é executado após todas as ações terem sido completadas na tabela especificada.


# Sintaxe para criação de um trigger em MySQL

Para criar um trigger em MySQL usamos a seguinte sintaxe:

- **CREATE TRIGGER nome timing operação**
- **ON tabela**
- **FOR EACH ROW**
- **declarações**
- **Onde:**
- **timing pode ser BEFORE ou AFTER**
- **operação pode ser INSERT / UPDATE / DELETE**

# Sintaxe para criação de um trigger em MySQL

Onde se tem os seguintes parâmetros:

**nome:** nome do gatilho, segue as mesmas regras de nomeação dos demais objetos do banco.

**momento:** quando o gatilho será executado. Os valores válidos são BEFORE **(antes)** e AFTER **(depois)**.

**evento:** evento que vai disparar o gatilho. Os valores possíveis são **INSERT, UPDATE** e **DELETE**. Vale salientar que os comandos **LOAD DATA** e **REPLACE** também disparam os eventos de inserção e exclusão de registros, com isso, os gatilhos também são executados.

**tabela:** nome da tabela a qual o gatilho está associado.

Não é possível criar mais de um trigger para o mesmo evento e momento de execução na mesma tabela. Por exemplo, não se pode criar dois gatilhos **AFTER INSERT** na mesma tabela.

# Os registros NEW e OLD

Como os triggers, são executados em conjunto com operações de inclusão e exclusão, é necessário poder acessar os registros que estão sendo incluídos ou removidos. Isso pode ser feito através das palavras **NEW** e **OLD**.

Em gatilhos executados após a inserção de registros, a palavra reservada **NEW** dá acesso ao novo registro. Pode-se acessar as colunas da tabela como atributo do registro **NEW**, como veremos nos exemplos.

O operador **OLD** funciona de forma semelhante, porém em gatilhos que são executados com a exclusão de dados, o **OLD** dá acesso ao registro que está sendo removido.

# Utilização do trigger

Para exemplificar e tornar mais clara a utilização de gatilhos, simularemos a seguinte situação: um mercado que, ao realizar vendas, precisa que o estoque dos produtos seja automaticamente reduzido. A devolução do estoque deve também ser automática no caso de remoção de produtos da venda.

Como se trata de um ambiente hipotético, teremos apenas duas tabelas de estrutura simples, cujo script de criação é mostrado abaixo:

Tabela-1:

```
CREATE TABLE Produtos(
Referencia     VARCHAR(3) PRIMARY KEY,
Descricao      VARCHAR(50) UNIQUE,
Estoque INT NOT NULL DEFAULT 0);
```

# Utilização do trigger

Inserção de valores:

```
INSERT INTO Produtos VALUES ("001", "Feijão", 10);
INSERT INTO Produtos VALUES ("002", "Arroz", 5);
INSERT INTO Produtos VALUES ("003", "Farinha", 15);
```

Tabela-2:

```
CREATE TABLE ItensVenda(
Venda         INT,
Produto VARCHAR(3),
Quantidade     INT);
```

# Explicando os passos

Ao inserir e remover registro da tabela **ItensVenda**, o estoque do produto referenciado deve ser alterado na tabela Produtos. Para isso, serão criados dois triggers: um **AFTER INSERT** para dar baixa no estoque e um **AFTER DELETE** para fazer a devolução da quantidade do produto.

***Observação: como usaremos instruções que requerem ponto e vírgula no final, alteraremos o delimitador de instruções para $$ e depois de criar os triggers, voltaremos para o padrão. Essa alteração não está diretamente ligada aos triggers.***

Faça um SELECT na tabela Produtos assim que terminar de inserir os valores.

# Criando os triggers com DELIMITER

```
DELIMITER $
CREATE TRIGGER Tgr_ItensVenda_Insert AFTER INSERT
ON ItensVenda
FOR EACH ROW
BEGIN
UPDATE Produtos SET Estoque = Estoque - NEW.Quantidade
WHERE Referencia = NEW.Produto;
END$
```

# Criando os triggers com DELIMITER

```
CREATE TRIGGER Tgr_ItensVenda_Delete AFTER DELETE
ON ItensVenda
FOR EACH ROW
BEGIN
UPDATE Produtos SET Estoque = Estoque + OLD.Quantidade
WHERE Referencia = OLD.Produto;
END$
DELIMITER ;
```

# Criando os triggers com DELIMITER

No primeiro gatilho, foi utilizado o registro **NEW** para obter as informações da linha que está sendo inserida na tabela. O mesmo é feito no segundo gatilho, onde se obtém os dados que estão sendo apagados da tabela através do registro **OLD**.

Tendo criado os triggers, podemos testá-los inserindo dados na tabela ItensVenda. Nesse caso, vamos simular uma venda de número 1 que contem três unidades do produto 001, uma unidade do produto 002 e cinco unidades do produto 003.

```
INSERT INTO ItensVenda VALUES (1, "001",3);

INSERT INTO ItensVenda VALUES (1, "002",1);

INSERT INTO ItensVenda VALUES (1, "003",5);
```

Apos ter efetuado todos os eventos, faça uma consulta na tabela de Produtos

Nota-se que o estoque dos produtos foi corretamente reduzido, de acordo com as quantidades “vendidas”.

Agora para testar o trigger da exclusão, removeremos o produto 001 dos itens vendidos. Com isso, o seu estoque deve ser alterado para o valor inicial, ou seja, 10.

```
DELETE FROM ItensVenda WHERE Venda = 1 AND Produto = "001";
```

Executando novamente um select na tabela Produtos, veremos que apenas o produto 001 teve o estoque atualizado, voltando a 10.

Com isso confirmamos que os gatilhos estão funcionando da forma esperada.

Abaixo temos um comando que exibe as triggers que foram criadas:

```
SHOW TRIGGERS
```

Excluusão de uma trigger no MySQL:

```
DROP TRIGGER Tgr_ItensVenda_Insert
```

# Informações adicionais

Apenas a nível de informação, vale o seguinte comentário: na nova versão do MySQL, no MySQL Workbench é possível visualizar os gatilhos relacionados a uma tabela através do Object browser.

Em ambientes reais, triggers podem ser utilizados para operações mais complexas, por exemplo, antes de vender um item, verificar se há estoque disponível e só então permitir a saída do produto.

Com isso finalizamos, onde foram apresentados os triggers no banco de dados MySQL.


# Mais um exemplo

Neste exemplo criaremos uma tabela chamada Produto, que conterá os seguintes dados:

- Nome do produto
- Identificação do produto (chave primária)
- Preço normal
- Preço com desconto a ser aplicado
- Logo após criaremos um trigger de nome tr_desconto, cuja função será aplicar um valor de desconto de 10% à coluna Preco_Desconto quando for disparado. Ou seja, todos os produtos terão seu preço reduzido em 10% nesta coluna. O trigger será disparado ao inserir um novo registro na tabela. Veja o código completo do exercício a seguir:


# Criando nosso primeiro exemplo

1. Criar a tabela de exemplo:

```
CREATE TABLE Produto (
idProduto INT NOT NULL AUTO_INCREMENT,
Nome_Produto VARCHAR(45) NULL,
Preco_Normal DECIMAL(10,2) NULL,
Preco_Desconto DECIMAL(10,2) NULL,
PRIMARY KEY (idProduto));
```


# Criando o trigger

```
CREATE TRIGGER tr_desconto BEFORE INSERT
ON Produto
FOR EACH ROW
SET NEW.Preco_Desconto = (NEW.Preco_Normal * 0.90);
```

Executar uma inserção que irá disparar o Trigger:

```
INSERT INTO Produto (Nome_Produto, Preco_Normal)
VALUES ("DVD", 1.00), ("Pendrive", 18.00);
```

Verificar se trigger foi disparado observando o preço com desconto:

```
SELECT * FROM Produto;
```


# Funcionamento do exemplo

Este exemplo, cria um acumulador somando os valores dentro das colunas especificas.

Para fazermos uso do gatilho, vamos setar o valor para **"0"**:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.01 sec)

mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
```